Homework 6, CSCE 240, Fall 2014

Overview

This is an exercise in using the `map` to organize information as keys and values, and to use the try-catch exception handling.

You are to read a simplified version of internet packet data and reassemble the packets into sorted order so that the underlying message can be read.

An internet packet consists of *metadata* and *payload*. The payload is the "data" part one actually sees. The metadata is the data that allows the packets to be sent individually through the net and to be put back in proper order as a complete message at the receiving end. This includes a message ID, a packet ID for each packet, the source IP address, the destination IP address, and a lot of other stuff as well.

For this assignment, the individual packets will have

1. A `messageID` that is unique for each message.
2. A `packetID` (numbered zero-up) that is unique for each packet and that is the sequence number of that packet in the overall ordering of packets for this message.
3. A `packetCount` value that is the number of packets in this entire message.
4. A `payload` that is the data for this particular message.

The first three of these are `int` variables; the last is a `string`.

Message processing thus works as follows. At the sending side, a message is given an ID, and then the message is broken into as many packets as are necessary to accommodate the size of the message.

(In the real internet, the payload has a fixed size. In this exercise, we will assume the payload is an instance of `string` data and can be of variable length.)

Packets are then sent out into the wilds of the net. They may take different routes to get from source to destination, so they may take various times in flight and arrive out of order.

The task of the receiving computer is to listen for packets. This will be done with a `PacketAssembler` class.

As the packets come in, the assembler will save them off, using a `map` to sort them automagically by `packetID`. When "the last" packet for a given message has arrived, the assembler with output the entire message (in ID order) and then delete the message.

The `Packet` class will be a simple class to store the data for a given packet, `toString` a packet, and such.

The `Message` class will be a class to store the data for a entire message, a message, and such. The basic data structure in the `Message` class will be a `map<int, Packet>` that uses the `packetID` as as key and by virtue of the `map` properties, keeps the packets so far encountered in sorted order by ID.

WE MAY ASSUME that each packet for a given message has the same value for `packetCount`, and thus that we can test for "we have all the packets" by checking to see if the size of the `map<int, Packet>` is equal to the value of `packetCount`.

Note, however, that packets can get sent multiple times, or can be sent in a garbled way and then resent, so the number of packets for a given message that have been received may well not be the same as the number of distinct packets that have been received.

YOU DO NOT NEED TO WORRY about whether it's the first packet received for a given `packetID` that is correct, or whether it's the last packet for that ID. In life, we don't have any good answer for that.

In other words, the packets will come in in no known order, but by storing them in a `map`, you will avoid the need to actually sort based on `packetID`. The search tree you create will allow you to read the packets back in sorted order by `packetID` simply by traversing the `map` in key order.

The `PacketAssembler` class will be a class that does most of the work of the program. The messages will come in in no known order, but the order by message ID will be kept by the `map<int, Message>` structure in the `PacketAssembler`.

Your code must handle duplicate instances of packets (duplicate by ID) and duplicate instances of messages. That is, if the complete set of packets for a given message comes in after the message has been completed, printed, and deleted, then your code should do all the work all over again, because it won't have saved any information to know that it has already seen that message.